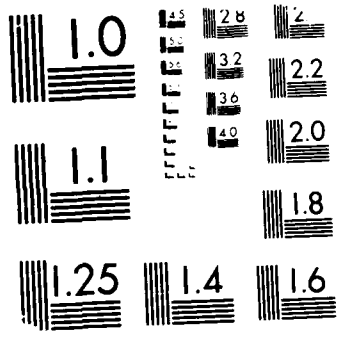
AD-A190 225 CONDITIONAL DESCRIPTIONS IN FUNCTIONAL UNIFICATION GRAMMAR(U) UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST R KASPER NOV 87 1/1

UNCLASSIFIED ISI-RR-87-191 F49620-87-C-0005 F/G 12/5 NL



ISI Research Report
ISI/RR-87-191
November 1987

Robert Kasper

Conditional Descriptions in Functional Unification Grammar

INFORMATION SCIENCES INSTITUTE
4676 Admiralty Way Marina del Rey, California 90292-6695

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT This document is approved for public release, distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/RR-87-191	5 MONITORING ORGANIZATION REPORT NUMBER(S) ----------
6a NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute 6b OFFICE SYMBOL *(If applicable)*	7a NAME OF MONITORING ORGANIZATION ----------
6c ADDRESS *(City, State, and ZIP Code)* 4676 Admiralty Way Marina del Rey, CA 90292	7b ADDRESS *(City, State, and ZIP Code)* ----------
8a NAME OF FUNDING / SPONSORING ORGANIZATION AFOSR 8b OFFICE SYMBOL *(If applicable)*	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-87-C-0005
8c ADDRESS *(City, State, and ZIP Code)* Air Force Office of Scientific Research Bolling Air Force Base, Building 410 Washington, DC 20332	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO ---------- PROJECT NO ---------- TASK NO. ---------- WORK UNIT ACCESSION NO. ----------

11 TITLE *(Include Security Classification)*
Conditional Descriptions in Functional Unification Grammar [Unclassified]

12 PERSONAL AUTHOR(S)
Kasper, Robert

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT *(Year, Month, Day)*	15 PAGE COUNT
Research Report	FROM _____ TO _____	1987, November	

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	artificial intelligence, computational linguistics, feature structures, grammar, logic, natural language, parsing, syntax, systemic grammar, unification grammar
09	02		

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Often, a grammatical description only applies to a linguistic object when that object has certain features. Such conditional descriptions can be indirectly modeled in Kay's Functional Unification Grammar (FUG) with functional descriptions that are embedded within disjunctive alternatives. An extension to FUG is proposed that allows for a direct representation of conditional descriptions. This extension has been used to model the input conditions on the systems of systemic grammar Conditional descriptions are formally defined in terms of logical implication and negation. This formal definition enables the use of conditional descriptions as a general notational extension to any of the unification-based grammar representation systems currently used in computational linguistics.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☑ UNCLASSIFIED/UNLIMITED ☑ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Sheila Coyazo, Victor Brown	22b TELEPHONE *(Include Area Code)* 213-822-1511 / 22c OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

ISI Research Report
ISI/RR-87-191
November 1987

University of Southern California

Robert Kasper

Conditional Descriptions in Functional Unification Grammar

INFORMATION SCIENCES INSTITUTE

213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by the Air Force Office of Scientific Research under Contract No. F49620-87-C-0005. Views and conclusions contained in this report are the author's and should not be interpreted as representing the official opinion or policy of AFOSR, the U.S. Government, or any person or agency connected with them.

Conditional Descriptions in Functional Unification Grammar *

Robert Kasper
USC / Information Sciences Institute

Abstract

Often, a grammatical description only applies to a linguistic object when that object has certain features. Such conditional descriptions can be indirectly modeled in Kay's Functional Unification Grammar (FUG) with functional descriptions that are embedded within disjunctive alternatives. An extension to FUG is proposed that allows for a direct representation of conditional descriptions. This extension has been used to model the input conditions on the systems of systemic grammar. Conditional descriptions are formally defined in terms of logical implication and negation. This formal definition enables the use of conditional descriptions as a general notational extension to any of the unification-based grammar representation systems currently used in computational linguistics.

1 Introduction

Functional Unification Grammar [6] (FUG) and other grammatical formalisms that use feature structures and unification provide a general basis for the declarative representation of natural language grammars. These formalisms provide a rich information structure that can express a wide variety of linguistic generalizations. A particularly attractive feature of these grammars is that they are readable by linguists and are also subject to interpretation by general computational tools, such as those for parsing or generating text. In order to exploit some of the computational tools available with unification grammars, we have developed a mapping from *systemic grammars* [1] into FUG notation. This mapping has been used as the first step in creating a general parsing method for systemic grammars [2]. The experience of translating systemic grammars into FUG has shown several ways in which the notational resources of FUG may be improved. In particular, FUG has limited notational resources for expressing conditional information.

This report describes how FUG has been enhanced by the addition of conditional descriptions, building on research that has already been reported [2,3,4]. In previous work, conditional descriptions were described informally as *selection conditions*. Selection conditions were formulated specifically to facilitate the modeling of systemic grammar, but the notion of conditional description developed in this report is more general.

*This research was sponsored by the United States Air Force Office of Scientific Research under contract F49620-87-C-0005; the opinions expressed here are solely those of the author.

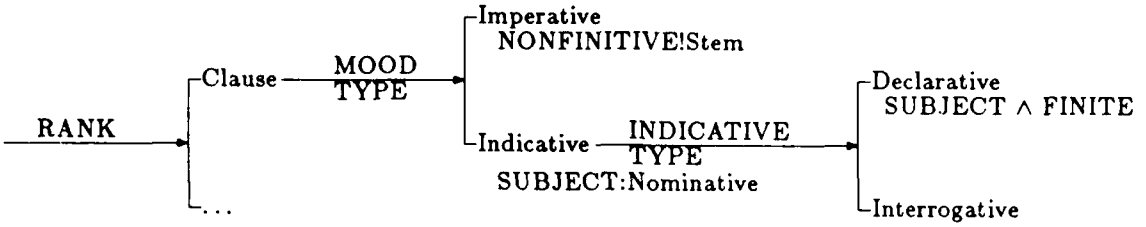


Figure 1: The MoodType and IndicativeType Systems

```
[ Rank = Clause
  ( [ [ MoodType = Imperative
        NONFINITIVE = [ Form = Stem ] ]
      [ MoodType = Indicative
        SUBJECT = [ Case = Nominative ]
        { [ IndicativeType = Declarative
            pattern = (... SUBJECT FINITE ...) ]
          [ IndicativeType = Interrogative ] } ] ] ) ]
```

Figure 2: The MoodType and IndicativeType Systems in FUG

2 Motivation: Conditional Information in Systemic Grammar

Conditional information is stated explicitly in systemic grammars by the input conditions of systems that specify when a system must be used. Consider, for example, the two systems (MoodType and IndicativeType) shown in Figure 1. The input condition for the MoodType system is the feature *Clause*, and the input condition for the IndicativeType system is the feature *Indicative*. Because the features of a systemic grammar are normally introduced by a unique system, these input conditions actually express a bidirectional type of logical implication:

1. If a constituent has the feature(s) specified by a system's input condition, then exactly one of the alternatives described by that system must also be valid for the constituent;

2. If a constituent has one of the feature alternatives described by a system, then it must also have the feature(s) specified by that system's input condition.

Thus the input condition of the *IndicativeType* system expresses the following implications:

1. If a clause has the feature *Indicative*, then it must also have exactly one of the alternatives from the *IndicativeType* system (either *Declarative* or *Interrogative*).

2. If a clause has one of the feature alternatives described by the *IndicativeType* system (either *Declarative* or *Interrogative*), then it must also have the feature *Indicative*.

While it is theoretically correct to regard the two directions of implication as exact converses of each other, there is a subtle difference between them. The consequent of the first type of implication is the description of the entire system, including systemic features and their

realizations.[1] The antecedent of the second type of implication can be safely abbreviated by the systemic features without their realizations, because the presence of a systemic feature implies that its realizations also hold. We will return to this distinction when we provide a formal definition of conditional descriptions in Section 3.

For simple input conditions, the first type of implication can be expressed in FUG, as it was originally formulated by Kay [6], by embedding the description of one system inside the description of another. For example, we can capture this implication for the IndicativeType system by embedding it within the description of the Indicative alternative of the MoodType system, as shown in Figure 2. Note that the second type of implication expressed by systemic input conditions has not been expressed by embedding one functional description inside another. To express the second type of implication, we have used a different notational device, called a feature existence condition; it will be defined in Section 3.4.

Not all systems have simple input conditions consisting of single features. Those input conditions which are complex boolean expressions over features cannot be expressed directly by embedding. Consider the DoingType system shown in Figure 3 as an example. Its input condition is the conjunction of two features, Effective and Material.

One way to express a system with a complex input condition in FUG is to use a disjunction with two alternatives, as shown in Figure 4. The first alternative corresponds to what happens when the DoingType system is entered; the second alternative corresponds to what happens when the DoingType system is not entered. The first alternative also includes the features of the input condition. The second alternative includes the features of the *negated* input condition. Notice that the input condition and its negation must both be stated explicitly, unlike in systemic notation. If the negation of the input condition was not included in the second alternative, it would be possible to use this alternative even when the input condition for the system holds. Thus the description of the system would not always be used when it should be. Note that this method of encoding systemic input conditions presupposes an adequate treatment of negated features. A formal definition of negation will be developed in Section 3.3.

While it is formally possible to encode complex input conditions by disjunction and negation, such encoding is not altogether satisfactory. It should not be necessary to state the negated input condition explicitly, since it can always be derived automatically from the unnegated condition. It is also rather inefficient to mix the features of the input condition with the other features of the system. The features of the input condition contain exactly the information that is needed to choose between the two alternatives of the disjunction (i.e., to choose whether the system is entered or not). It would be more efficient and less verbose to have a notation in which the features of the input condition are distinguished from the other features of the system, and in which the negation of the input condition does not need to be stated explicitly. Therefore, we have developed an extension to FUG that uses a conditional operator ($\longleftrightarrow$), as illustrated by the encoding of the DoingType system shown in Figure 5. A description corresponding to the input condition appears to the left of the $\longleftrightarrow$ symbol, and the description to be included when the input condition is satisfied appears to its right. Note that using a bidirectional condition allows us to capture both of the logical implications that are entailed by an input condition in systemic grammar. A formal definition of what it means for a description to be satisfied will be given in Section 3.1.

[1] A realization is a statement of structural properties that are required by a feature, such as the statement that SUBJECT precedes FINITE for the feature declarative.

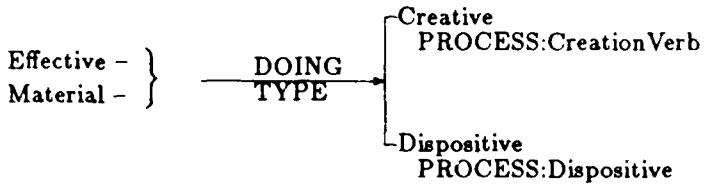


Figure 3: The DoingType System

```
[ Rank = Clause
  ( { [ Agency = Effective
        ProcessType = Material
        { [ DoingType = Creative
            PROCESS = [ VerbType = CreationVerb ] ]
          [ DoingType = Dispositive
            PROCESS = [ VerbType = Dispositive ] ] } ]
      [ { [ Agency = NOT Effective ]
          [ ProcessType = NOT Material ] }
        DoingType = none ] } ]
```

Figure 4: DoingType system in FUG, using disjunction and negation.

```
[ Rank = Clause
  [ Agency = Effective
    ProcessType = Material ] <-> { [ DoingType = Creative
                                     PROCESS = [ VerbType = CreationVerb ] ]
                                   [ DoingType = Dispositive
                                     PROCESS = [ VerbType = Dispositive ] ] } ]
```

Figure 5: DoingType system in FUG, simplified by using a conditional description.

Note that in systemic notation curly braces represent conjunction and square braces represent disjunction, while in FUG curly braces represent disjunction and square braces represent conjunction.

NIL	denoting *no information*;
TOP	denoting *inconsistent information*;
a	where $a \in A$, to describe atomic values;
$l : \phi$	where $l \in L$ and $\phi \in$ FDL, to describe structures in which the feature labeled by l has a value described by ϕ;
$[\![< p_1 >, \ldots, < p_n >]\!]$	where each $p_i \in L^*$, to describe an equivalence class of paths sharing a common value in a feature structure;
$\phi_1 \wedge \phi_2$ or $[\phi_1 \ldots \phi_n]$	where $\phi_i \in$ FDL, denoting conjunction;
$\phi_1 \vee \phi_2$ or $\{\phi_1 \ldots \phi_n\}$	where $\phi_i \in$ FDL, denoting disjunction.

Note: A and L are sets of symbols which are used to denote atomic values and feature labels, respectively.

Figure 6: Syntax of FDL Formulas.

3 Definitions

The feature description logic (FDL) of Kasper and Rounds [3] provides a coherent framework to give a precise interpretation for conditional descriptions. As in previous work, we carefully observe the distinction between feature structures and their descriptions. Feature structures are represented by directed graphs (DGs), and descriptions of feature structures are represented by logical formulas. The syntax for formulas of FDL is given in Figure 6. We define several new types of formulas for conditional descriptions and negations, but the domain of feature structures remains DGs, as before.

3.1 Satisfaction and Compatibility

In order to understand how conditional descriptions are used, it is important to recognize two relations that may hold between a particular feature structure and a description: satisfaction and compatibility. Satisfaction implies compatibility, so there are three possible states that a description may have with respect to a particular structure:

1. the description may be fully ***satisfied*** by the structure;
2. the description may be ***compatible*** with (but not satisfied by) the structure;
3. the description may be ***incompatible*** with the structure.

To define these terms more precisely, we say that an atomic feature description, $f : v$, is:

satisfied by A if f occurs with value v in A;

(merely) compatible with A if f does not occur in A, and any feature existence conditions for f are compatible with A;

incompatible with A otherwise, i.e., if f occurs with value x in A, for some $x \neq v$, or if f has a feature existence condition which is incompatible with A.

Logical combinations of feature descriptions are evaluated with their usual semantics to determine whether they are satisfied or compatible with a structure. Thus, a conjunction is satisfied only when every conjunct is satisfied, and a disjunction is satisfied if any disjunct is satisfied.

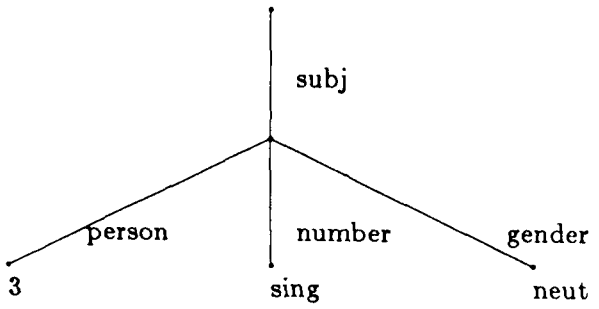


Figure 7: Example feature structure ($\mathcal{A}$).

Consider, for example, the structure ($\mathcal{A}$) shown in Figure 7, and the three descriptions:

$$\textit{subj} : (\textit{person} : 3 \wedge \textit{number} : \textit{sing}) \quad (1)$$

$$\textit{subj} : (\textit{person} : 1 \wedge \textit{number} : \textit{sing}) \quad (2)$$

$$\textit{subj} : (\textit{case} : \textit{nom} \wedge \textit{number} : \textit{sing}) \quad (3)$$

Description (1) is *satisfied* by $\mathcal{A}$, because $\mathcal{A}$ is fully instantiated with all the required feature values. Description (2) is *incompatible* with $\mathcal{A}$, because $\mathcal{A}$ has a different value for the feature $\textit{subj} : \textit{person}$. Description (3) is *merely compatible* with $\mathcal{A}$ (but not satisfied by $\mathcal{A}$), because $\mathcal{A}$ has no value for the feature $\textit{subj} : \textit{case}$. Because feature structures are used to represent partial information, it is possible for $\mathcal{A}$ to be extended (i.e., by adding a value for the feature $\textit{subj} : \textit{case}$) so that it either satisfies or becomes incompatible with description (3).

In the following definitions, the notation $\mathcal{A} \models \phi$ means that the structure $\mathcal{A}$ *satisfies* the description ϕ.

3.2 Conditional Description

We augment FDL with a new type of formula to represent conditional descriptions, having the syntax:

$$\alpha \rightarrow \beta$$

and the interpretation:

$$\mathcal{A} \models \alpha \rightarrow \beta \iff \mathcal{A} \models \neg\alpha \vee \beta. \quad (4)$$

We often refer to formulas of this type simply as conditionals.

Our definition of conditional descriptions is formally equivalent to material implications. This interpretation of conditionals presupposes an interpretation of logical negation, which is given below. To simplify the interpretation of negations, we exclude formulas containing path equivalences and path values from the antecedents of conditionals.

A bidirectional condition operator ($\leftrightarrow$) is also used with its normal interpretation:

$$\begin{aligned} \alpha \leftrightarrow \beta &\iff \alpha \rightarrow \beta \wedge \beta \rightarrow \alpha \\ &\iff (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta). \end{aligned}$$

3.3 Negation

We use the classical interpretation of negation, where $\mathcal{A} \models \neg\phi \iff \mathcal{A} \not\models \phi$. Negated descriptions are defined for the following types of formulas:

1. $\mathcal{A} \models \neg a \iff \mathcal{A}$ is not the atom a;
2. $\mathcal{A} \models \neg(l : \phi) \iff \mathcal{A} \models l : \neg\phi$ or $\mathcal{A}/l$ is not defined;
3. $\mathcal{A} \models \neg(\phi \vee \psi) \iff \mathcal{A} \models \neg\phi \wedge \neg\psi$;
4. $\mathcal{A} \models \neg(\phi \wedge \psi) \iff \mathcal{A} \models \neg\phi \vee \neg\psi$.

Note that we have not defined negation for formulas containing path equivalences or path values. This restriction makes it possible to reduce all occurrences of negation to a boolean combination of a finite number of negative constraints on atomic values. While the classical interpretation of negation is not strictly monotonic with respect to the normal subsumption ordering on feature structures, the restricted type of negation proposed here does not suffer from the inefficiencies and order-dependent unification properties of general negation or intuitionistic negation [7,8]. The reason for this is that we have restricted negation so that all negative information can be specified as local constraints on single atomic values.

3.4 Feature Existence Conditions

Conditional descriptions have been defined as an implication relation between two feature descriptions. A slightly different type of conditional description is needed when the antecedent of the conditional is an existence predicate for a particular feature, and not a regular feature description. We call this type of conditional a *feature existence condition*. Feature existence conditions are needed to model the second type of implication expressed by systemic input conditions – namely, when a constituent has one of the feature alternatives described by a system, it must also have the feature(s) specified by that system's input condition. A *feature existence condition* can be stated formally as:

$$\exists f \rightarrow \phi,$$

where $\mathcal{A} \models \exists f \iff \mathcal{A}/f$ is defined. This use of $\exists f$ is essentially equivalent to the use of $f = ANY$ in FUG, where ANY is a place-holder for any substantive (i.e., non-NIL) value.

The antecedent ($\exists f$) might also be described by the FDL formula $f : NIL$, however this notation could be confusing. Using the formal interpretation that we have given for formulas of the type $l : \phi$, we can derive

$$\mathcal{A} \models f : NIL \iff \mathcal{A} \models \exists f,$$

since $\mathcal{A} \models f : NIL \iff \mathcal{A}/f$ is defined and $\mathcal{A}/f \models NIL$, and any value for the feature f satisfies NIL. The confusion in the interpretation of $f : NIL$ arises because some implementations of unification for feature structures, such as PATR-II [9], have often freely introduced features with the value NIL.[2] In these implementations $f : NIL$ has been

[2] NIL values are introduced when two paths in a feature structure are unified and one or both of those paths does not yet exist. When one of the paths already has a value, then introducing a NIL value for the other path simply allows the unification to succeed in all cases, and the NIL value is replaced by a substantive value during unification. When two nonexistent paths are unified, a NIL value remains in the structure after unification, indicating that nothing is known about the value of the paths that have been unified. These NIL values are usually regarded as unbound variables, not necessarily implying the existence of any substantive value for the paths that have been unified.

defined to be equivalent to NIL out of convenience, because the implementors did not have any need to state feature existence conditions.

The primary effect of a feature existence condition, such as $\exists f \rightarrow \phi$, is that the consequent is asserted whenever a substantive value is introduced for a feature labeled by f. The treatment of feature existence conditions differs slightly from other conditional descriptions in the way that an unsatisfiable consequent is handled. In order to negate the antecedent of $\exists f \rightarrow \phi$, we need to state that f may never have any substantive value. This is accomplished by unifying a special atomic value, such as $NONE$, with the value of f. This special atomic value is incompatible with any other real value that might be proposed as a value for f.

4 Unification with Conditional Descriptions

The unification operation, which is commonly used to combine feature structures (i.e., non-disjunctive, non-conditional DGs), can be generalized to define an operation for combining the information of two feature descriptions (i.e., formulas of FDL). In FDL, the unification of two descriptions is equivalent to their logical conjunction, as discussed in [4]. We have shown in previous work [5] how unification can be accomplished for disjunctive descriptions.

This unification method factors descriptions into definite and indefinite components. The definite component contains no disjunction, and is represented by a DG structure that satisfies all non-disjunctive parts of a description. The indefinite component of a description is a list of disjunctions. When two descriptions are unified, the first step is to unify their definite components. Then the indefinite components of each description are checked for compatibility with the resulting definite component. Disjuncts are eliminated from the description when they are inconsistent with definite information. When only one alternative of a disjunction remains, it is unified with the definite component of the description.

This section details how this unification method can be extended to handle conditional descriptions. Conditionals may be regarded as another type of indefinite information in the description of a feature structure. They are indefinite in the sense that they impose constraints that can be satisfied by several alternatives, depending on the values of features already present in a structure.

4.1 How to Satisfy a Conditional Description

Let us focus on how a conditional description can be used to impose constraints on a feature structure. The constraints imposed on a feature structure by a conditional description can usually be determined most efficiently by first examining the antecedent of the conditional, because it generally contains a smaller amount of information than the consequent. Examining the antecedent is often sufficient to determine whether the consequent is to be included or discarded.

Given a conditional description, $C = \alpha \rightarrow \beta$, we can define the constraints that it imposes on a feature structure (A) as follows. When A:

satisfies α, then $A \models \beta$;[3]

is incompatible with α, then C imposes no further constraint on A, and can therefore be eliminated;

[3] Read this constraint as: "make sure that A satisfies β."

is merely compatible with α, then check whether β is compatible with $\mathcal{A}$.

If compatible, then C must be retained in the description of $\mathcal{A}$.

If incompatible, then $\mathcal{A} \models \neg\alpha$ (and C can be eliminated).

These constraints follow directly from the interpretation (4) that we have given for conditional descriptions. These constraints are logically equivalent to those that would be imposed on $\mathcal{A}$ by the disjunction $\neg\alpha \vee \beta$, as required by our definition. However, the constraints of the conditional can often be imposed more efficiently than those of the equivalent disjunction, because examining the antecedent of the conditional carries the same cost as examining only one of the disjuncts. When the constraints of a disjunction are imposed, both of the disjuncts must be examined in all cases.

4.2 Extending the Unification Algorithm

The unification algorithm for disjunctive feature descriptions [5] can be extended to handle conditionals by recognizing two types of indefinite information in a description: disjunctions and conditionals. The extended *feature-description* data structure has the components:

definite: a DG structure;

disjunctions: a list of disjunctions;

conditionals: a list of conditional descriptions.

The part of the unification algorithm that checks the compatibility of indefinite components of a description with its definite component is defined by the function CHECK-INDEF, shown in Figure 8.

An equally correct version of this algorithm might check conditionals before checking disjunctions. In our application of parsing with a systemic grammar it is generally more efficient to check disjunctions before conditionals, but other applications might be made more efficient by varying this order.

5 Representing Systemic Grammars by Conditional Descriptions

In Section 2 we introduced conditional descriptions by showing how the systems of systemic grammars can be represented by a type of bidirectional condition. In practice it is useful to decouple the two directions of implication so that they can be applied at different times. Generally, a system named f with input condition α and alternatives described by β, can be represented by two conditional descriptions:

1. $\alpha \rightarrow \beta$;
2. $\exists f \rightarrow \alpha$.

Note that the second conditional is not an exact converse of the first. A feature existence condition can be used as the antecedent of the second conditional instead of the entire description of the alternatives of the system (β). For example, the DoingType system, originally shown in Figure 3, is represented in FDL by the two conditional descriptions shown in Figure 9.

It is important to note that the second type of formula used in this translation – a feature existence condition – is used for systems with simple input conditions as well as for

Function CHECK-INDEF (desc) *Returns* feature-description:
where desc is a feature-description.

Let $\mathcal{D}$ = desc.definite (a DG).
Let disjunctions = desc.disjunctions.
Let conditionals = desc.conditionals.
Let unchecked-parts = true.

While unchecked-parts, do:

unchecked-parts := false.

Check compatibility of disjunctions with $\mathcal{D}$.

Check compatibility of conditionals with $\mathcal{D}$.
Let new-conditionals = $\emptyset$.

For each $\alpha \rightarrow \beta$ in conditionals:
test whether $\mathcal{D}$ satisfies or is compatible with α:
SATISFIES: $\mathcal{D}$:= UNIFY-DGS ($\mathcal{D}$, β.definite),
disjunctions := disjunctions $\cup$ β.disjunctions,
unchecked-parts := true;
COMPATIBLE: If β is compatible with $\mathcal{D}$,
then new-conditionals := new-conditionals $\cup$ $\{\alpha \rightarrow \beta\}$,
else let neg-ante = $\neg\alpha$,
$\mathcal{D}$:= UNIFY-DGS ($\mathcal{D}$, neg-ante.definite),
disjunctions := disjunctions $\cup$ neg-ante.disjunctions,
unchecked-parts := true;
INCOMPATIBLE: *this conditional imposes no further constraint.*
end (for loop).

conditionals := new-conditionals.
end (while loop).

Let new-desc = make feature-description with:
new-desc.definite = $\mathcal{D}$,
new-desc.disjunctions = disjunctions,
new-desc.conditionals = conditionals.

Return (new-desc).

Figure 8: Algorithm to check compatibility of indefinite parts of a feature-description.

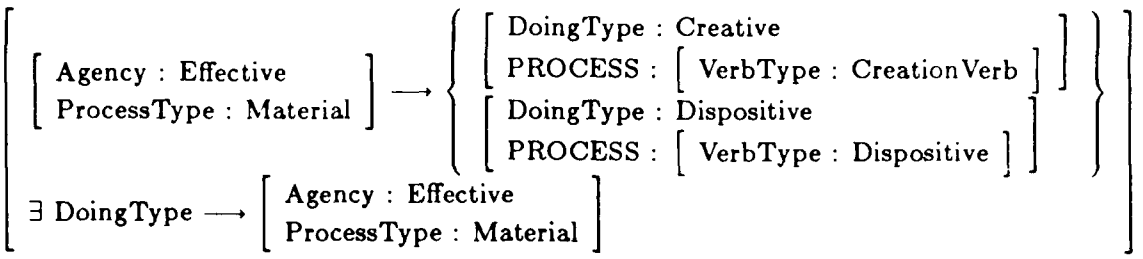


Figure 9: Translation of DoingType system, using two conditional descriptions.

those with complex input conditions. The use of the feature existence condition is essential in both cases to encode the bidirectional dependency between systems that is implicit in a systemic network. In our implementation of a parser, feature existence conditions are checked in a more efficient manner than other conditional descriptions: a feature existence condition with $\exists f$ as its antecedent only needs to be checked immediately after a feature with label f is added to a structure.

6 Potential Refinements

Several topics require further investigation regarding conditional descriptions. The implementation we describe has the constraints of conditionals and disjunctions imposed in an arbitrary order. Changing the order has no effect on the final result, but it is likely that the efficiency of unification could be improved by ordering the conditionals of a grammar in a deliberate way. Another way to improve the efficiency of unification with conditionals would involve indexing them by the features that they contain. Then a conditional would not need to be checked until some feature value determines whether it is satisfied. The amount of efficiency gained by such techniques clearly depends largely on the nature of the particular grammar being used in an application.

A slightly different type of conditional might be used as a way to speed up unification with binary disjunctive descriptions. If it is known that the values of a relatively small number of features can be used to discriminate between two alternative descriptions, then those features can be factored into a separate condition in a description such as

$$\text{IF } \textit{condition} \text{ THEN } alt_1 \text{ ELSE } alt_2.$$

When the condition is satisfied by a structure, then alt_1 is selected. When the condition is incompatible with a structure, then alt_2 is selected. Otherwise both alternatives must remain under consideration. As it often requires a considerable amount of time to check which alternatives of a disjunction are applicable, this technique might offer a significant improvement in an application where large disjunctive descriptions are used.

Remember that we have restricted conditionals by requiring that their antecedents do not contain path equivalences. This restriction has been acceptable in our use of conditional descriptions to model systemic grammars. It is unclear whether a treatment of conditional descriptions without this restriction will be needed in other applications. If this restriction is lifted, then further work will be necessary to define the behavior of negation over path equivalences, and to handle such negations in a reasonably efficient manner.

7 Summary

We have shown how the notational resources of FUG can be extended to include descriptions of conditional information about feature structures. Conditional descriptions have been given a precise logical definition in terms of the feature description logic of Kasper and Rounds, and we have shown how a unification method for feature descriptions can be extended to use conditional descriptions. We have implemented this unification method and tested it in a parser for systemic grammars, using several hundred conditional descriptions. The definition of conditional descriptions and the unification method should be generally applicable as an extension to other unification-based grammar frameworks, as well as to FUG and the modeling of systemic grammars. In fact, the implementation described has been carried out by extending PATR-II [9], a general representational framework for unification-based grammars.

It is theoretically possible to represent the information of conditional descriptions using several notational devices already present in Kay's FUG. Conditionals descriptions can be represented by disjunctions, as we have shown in Figure 4, and feature existence predicates and their negations can be represented using the special values *ANY* and *NONE*. Although theoretically possible, encoding conditional descriptions by disjunctions entails approximately doubling the size of the description and slowing the unification process. Therefore, by adding conditional descriptions, we have not changed the theoretical limits of what FUG can do, but we have developed a representation that is more perspicuous, less verbose, and computationally more efficient.

References

[1] G.R. Kress, editor. *Halliday: System and Function in Language*. Oxford University Press, London, England, 1976.

[2] Kasper, R. Systemic Grammar and Functional Unification Grammar. In J. Benson and W. Greaves, editors, *Systemic Functional Perspectives on Discourse: Selected Papers from the 12th International Systemics Workshop*, Norwood, New Jersey: Ablex (forthcoming). Also available as USC/Information Sciences Institute, Technical Report RS-87-179, May 1987.

[3] Kasper, R. and W. Rounds. A Logical Semantics for Feature Structures. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, NY, June 10-13, 1986.

[4] Kasper, R. *Feature Structures: A Logical Theory with Application to Language Analysis*. PhD dissertation, University of Michigan, 1987.

[5] Kasper, R. A Unification Method for Disjunctive Feature Descriptions. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, CA, July 6-9, 1987.

[6] Kay, M. Functional Grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, Berkeley Linguistics Society, Berkeley, California, February 17-19, 1979.

[7] Moshier, M. D. and W. C. Rounds. A Logic for Partially Specified Data Structures. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, 1987.

[8] Pereira, F.C.N. Grammars and Logics of Partial Information. In *Proceedings of the International Conference on Logic Programming*, Melbourne, Australia, May 1987.

[9] Shieber, S. M. The design of a computer language for linguistic information. In *Proceedings of the Tenth International Conference on Computational Linguistics: COLING 84*, Stanford University, Stanford, California, July 2-7, 1984.

END

DATE

FILMED

4-88

DTIC